Hadoop Components: MapReduce and HDFS

Hadoop Components, MapReduce and HDFS.
Our learning objectives for this video are discovering the key elements of both MapReduce and HDFS, and the functionality that their components bring to the Hadoop framework.
A Task Tracker is a node in the cluster that accepts tasks, such as MapReduce and shuffle operations from the job tracker.
Every task tracker is configured with a set of slots that indicate the number of tasks that it can accept.
When a job tracker tries to find somewhere to schedule a task within the MapReduce operations, it first looks for an empty slot on the same server that hosts the data node containing the data.
And, if not, it looks for an empty slot on a machine in the same rack.
Task Tracker spawns separate JVM processes to do the actual work and ensure that process failure does not take it down.
TaskTracker monitors these spawned processes, capturing output and exit codes.
When the process finishes successfully or not, the tracker notifies the JobTracker.
TaskTracker also sends out heartbeat messages to JobTracker, usually every few minutes to reassure the JobTracker that it is still alive.
These messages also inform the JobTracker of the number of available slots so the JobTracker can stay up-to-date with where in the cluster work can be delegated.
Let's go over the main concepts for MapReduce.
Hadoop MapReduce is programming and run-time framework for analyzing data sets stored in HDFS.
It's designed the way you write applications that rapidly process vast amounts of data in parallel on a large cluster of compute nodes.
MapReduce jobs are composed of user-supplied Map and and Reduce functions.
Map subdivides tasks into smaller subtasks and then it sends across various reducers.
Reduce combines the subtasks and reduced cardinality.
Map functions take a divide and conquer approach to processing data as fast as possible.
Reduce function and reduce combine and reduce cardinality.
The MapReduce framework provides all the glue and coordinates that the execution of the maps and reduce jobs on the cluster, map function take a large problem and divide it into subproblems and perform the same function on all sub-problems.

Reduce functions perform the combined process,
which takes the output from all sub-problems and
creates a result set out of it.
Often this output is in the form of a delimited text file that
can be easily consumed by the speed and serving layers.
A MapReduce job splits a large data set into
individual chunks and
organizes them into key value pairs for parallel processing.
This parallel processing improves the speed and
reliability of the cluster,
returning solutions more quickly and with greater reliability.
The map function divides the input into ranges by the input
format and created a map task for each range of the input.
The JobTracker then distributes those task to the worker nodes.
The output of each map task is partitioned into a group of
key value pairs for each reduced.
The reduced function then collects the various results and
combines them to answer the larger problem that the master
node needs to solve.
Each reduce pulls the relevant partition from the machines
where the map's executed,
then writes its output back into HDFS.
Thus, the reduce is able to collect the data
from all the maps for the keys and
combine them to solve the problem.
The Hadoop Distributed File System that we've been talking
about, which is HDFS,
is a subproject of the Apache Hadoop project.
This Apache Software Foundation project is designed to
provide a fault-tolerant file system
designated to run on commodity hardware.
HDFS has many similarities with existing distributed
file systems.
However, the differences from other distributed file systems
are significant.
HDFS is highly fault tolerant and
is designed to be deployed on low cost hardware.
HDFS provides high throughput access to application data and
is suitable for applications that have large data sets.
HDFS relaxes a few POSIX requirements to enable streaming
access to file system data.
HDFS is designed to reliably store very large files across
machines in a large cluster.
It stores each file as a sequence of blocks with
all blocks in a file, except the last block, being the same size.
The blocks of a file are replicated for fault tolerance.

Block size and
replication factor are configurable per file.
An application can specify the number of replicas of a file.
The replication factor can be specified at file creation time
and can be changed later.
Files in HDFS are write-once and
have strictly one writer at any time.
HDFS is a master at detail kind of architecture.
Name node acts as a master and data nodes as a worker.
All the metadata information is with name node and
the original data is stored on the data nodes.
This figure gives an idea about how data flow happens between
the clients interacting with HDFS, or
the name nodes and data nodes.
The following steps are involved in reading the file from HDFS.
Let's suppose a client, an HDFS client,
wants to read a file from HDFS.
The first step is that the client will open the file by
giving a call to the open method on the file system object
which for HDFS is an instance of distributed file system class.
Step two, the distributed file system calls the name node using
RPC, which is remote procedure call, to determine the locations
for the blocks for the first few blocks of the file.
For each block, the name node returns the address of
all the data nodes that will have a copy of that block.
The client will interact with respective data nodes to read
the file.
Step three, the client then calls read on the stream.
DFS input stream which has stored the data node
addresses for the first few blocks in the file then connects
to the first closest data node for the first block in the file.
Step four, the data is streamed from the data node
back to the client, which calls read repeatedly on that stream.
And in the fifth step, when the end of the block is reached,
DFS input stream will close the connection to the data node and
then find the best data node for the next block.
Step six, the blocks are read in order
with the DFS input stream opening new connections to
data nodes as the client reads through the stream.
It will also call the name node to retrieve the data node
locations for the next batch of blocks as needed.
When the client has finished reading,
it calls close on the FS data input stream.
In writng data to HDFS, there are sevens steps.
Step one, the client creates the file by calling
the create method on the distributed file system.

Step two, the distributed file system makes
an RPC call to the name node to create a new file in the file
systems' namespace with no blocks associated with it.
Step three, as the client writes data,
DFS output stream splits it into packets which it then writes
to an internal queue, called the data queue.
The data queue is consumed by the data streamer which is
responsible for asking the name node to allocate new blocks by
picking a list of suitable data nodes to store the replicas.
Step four, similarly, the second data node stores the packet and
forwards it to the third and last data node in the pipe line.
Step five, DFS output stream also maintains an internal queue
of packets that are waiting to be acknowledged by data nodes,
called an ACK queue.
The packet is removed from the ACK queue only when it has
acknowledged by all the data nodes in the pipeline.
Step six, when the client has finished writing data,
it calls close on the stream.
In the final step, seven, this action flushes
all the remaining packets to the data node pipeline and waits for
acknowledgements before contacting the name node
to signal that the file is complete.
As previously stated, HDFS has a master detail architecture.
An HDFS cluster consists of a single NameNode,
a master server that manages the file system namespace and
regulates access to the files by clients.
In addition, there are a number of data nodessually
one per node in the cluster,
which manage storage attached to the nodes that they run on.
HDFS exposes the file system namespace, and
allows user data to be stored in files.
The name node actively monitors data node health, and
the number of replicas in a block.
When a replica of a block is lost due to a data node
failure or disk failure,
the NameNode creates another replica of the block.
The NameNode then maintains the namespace tree and
the mapping of blocks to data nodes,
holding the entire namespace image in RAM.
Internally, a file is split into one or more blocks, and
these blocks are stored in a set of data nodes.
The NameNode executes file system name space operations,
like opening, closing and renaming files and directories.
It also determines the mapping of blocks to data nodes.
The data nodes are responsible for serving read and
write requests from the file system's clients.

The data node performs the following.
It manages data blocks and serves them to clients.
There are many per Hadoop cluster controlled by NameNode.
It periodically reports to NameNode the list of
blocks that it stores.
You should use inexpensive commodity hardware for
this node because when it fails NameNode, has options for
replacements available.